

# Cyclic Operator Precedence Grammars for Parallel Parsing<sup>\*</sup>

Michele Chiari<sup>a</sup>, Dino Mandrioli<sup>b</sup>, Matteo Pradella<sup>b,c</sup>

<sup>a</sup>*Institute of Computer Engineering, TU Wien, Treitlstraße 3, 1040, Vienna, Austria*

<sup>b</sup>*Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133, Milano, Italy*

<sup>c</sup>*IEIIT, Consiglio Nazionale delle Ricerche, via Ponzio 34/5, 20133, Milano, Italy*

---


## Abstract

Operator precedence languages (OPLs) enjoy the local parsability property, which means that a code fragment enclosed within a pair of markers playing the role of parentheses can be parsed with no knowledge of its external context. This property has been exploited to build parallel parsers for languages formalized as OPLs. It has been observed, however, that when the syntax trees of sentences have a linear substructure, parsing must necessarily proceed sequentially, making it ineffective to split such a subtree into chunks to be processed in parallel. This inconvenience derives from the hypothesis that the equality precedence relation cannot be cyclic, which has been so far assumed by most literature on OPLs. This hypothesis was motivated by the need to keep the mathematical notation as simple as possible, although it caused a discrepancy between the expressive power of operator precedence grammars and other formalisms defining OPLs such as operator precedence automata, monadic second order logic and operator precedence expressions, which do not assume acyclicity.

We present an enriched version of operator precedence grammars, called *cyclic*, that allows for a simplified version of regular expressions in the right hand sides of grammar rules. For this class of operator precedence grammars the acyclicity hypothesis of the equality precedence relation is no more needed to guarantee the algebraic properties of the generated languages. The expressive power of the cyclic grammars is now fully equivalent to that of other formalisms defining OPLs. As a result, cyclic operator precedence grammars produce unranked syntax trees and sentences with flat unbounded substructures that can be naturally partitioned into chunks suitable for parallel parsing.

*Keywords:* Operator Precedence Languages, Cyclic Precedence Relations, Parallel Parsing

---

<sup>\*</sup>This work was partially funded by the European Commission  within the Horizon Europe programme under Grant No. 101107303 “CORPORA”.

*Email addresses:* [michele.chiari@tuwien.ac.at](mailto:michele.chiari@tuwien.ac.at) (Michele Chiari),  
[dino.mandrioli@polimi.it](mailto:dino.mandrioli@polimi.it) (Dino Mandrioli), [matteo.pradella@polimi.it](mailto:matteo.pradella@polimi.it) (Matteo Pradella)

---

## 1. Introduction

*Operator precedence languages (OPLs)* are a “historical” family of languages invented by R. Floyd [1] to support fast deterministic parsing. Together with their *operator precedence grammars (OPGs)*, they are still used within modern compilers to parse expressions with operators ranked by priority. The key feature that makes them well amenable for efficient parsing and compilation is that the syntax tree of a sentence is determined exclusively by three binary precedence relations over the terminal alphabet that are easily pre-computed from the grammar productions. For readers unacquainted with OPLs, we anticipate an example: the arithmetic sentence  $a + b \times c$  does not make manifest the natural structure  $(a + (b \times c))$ , but the latter is implied by the fact that the plus operator yields precedence to the times.

Early theoretical investigation [2], originally motivated by grammar inference goals, realized that, thanks to the tree structure assigned to strings by the precedence relations, many closure properties of regular languages hold for OPLs too, but only after a long intermission, renewed research [3] proved further algebraic properties of OPLs. Then, Lonati et al. [4] defined *Operator Precedence automata (OPAs)* as a formalism, paired with OPGs as well as pushdown automata are paired with context-free grammars (CFGs), to formalize the recognition and parsing of OPLs. In the same paper, a *monadic second-order (MSO) logic* characterization of OPLs that naturally extends the classic one for regular languages was also produced. Furthermore, Mandrioli et al. [5] introduced the *operator precedence expressions (OPEs)*, which extend traditional regular expressions in the same way as the MSO logic for OPLs extends the traditional one for regular languages. The extension of fundamental algebraic and logic properties of regular languages to OPLs has been completed by Henzinger et al. [6] where a characterization of OPLs in terms of a syntactic congruence with a finite number of equivalence classes is given.

Another distinguishing property of OPLs is their *local parsability*, i.e., the fact that a chunk of text included within a pair of symmetric precedence relations can be deterministically parsed even without knowing its context [7]. This feature was exploited to produce a parallel parser generator which exhibited high performances w.r.t. traditional sequential parsers [8].

A different research branch—which is not the core aspect of this paper—exploited the algebraic and logic properties of this family to extend to it the successful verification techniques based on model checking, with comparable performances with those obtained for regular languages [9]. In summary, this historical family of languages enjoys properties that allowed relevant modern applications such as automatic verification and parallel compilation.

It must be pointed out, however, that many, but not all, of the algebraic properties discovered during such a research activity were proved by assuming a hypothesis on the precedence relations defined on the input alphabet. From a theoretical point of view, this hypothesis slightly affects the generative power of

OPGs but is not necessary, e.g., for OPAs and MSO logic: these two formalisms are thus a little more powerful than OPGs. So far, no practical limitation due to it was discovered in terms of formalizing the syntax of real-life programming and data description languages. Thus, it has been constantly adopted in the various developments to avoid making the mathematical notation and technical details too cumbersome.

Li and Taura [10], however, recently pointed out a weakness of the parallel compilation algorithm described in [8], which in some cases hampers partitioning the input to be parsed in well-balanced chunks, thus diminishing the benefits of parallelism. Intuitively, the weakness is due to the fact that the “normal” precedence relation on the arithmetic operator  $+$  compels to parse a sequence thereof by associating them either to the left or to right so that parsing becomes necessarily sequential. The authors also proposed a special technique to overcome this difficulty by allowing for an acceptable level of ambiguity, which in the case of OPGs determines a conflict for some precedence relations on the terminal alphabet.

The “traditional” precedence relation, however, which either let  $+$  yield precedence to, or take precedence over itself, has no correspondence with the semantics of the addition operation, whose result does not depend on the order of association. So, why not giving the various occurrences of the  $+$  operator the same precedence level as suggested by arithmetic’s laws?<sup>1</sup> Figure 1 gives an intuitive idea of the different structures given to a sequence of  $+$  operators by traditional OPGs vs. the natural semantics of the sum operation. Whereas the left syntax tree can be built by a bottom-up parser only in a strictly sequential order, we can imagine that two parallel processors could build independently portions of the right tree that can be integrated in a following pass. More precise explanations will be given in the following sections.

The answer to this question comes exactly from the above mentioned hypothesis: it forbids cyclic sequences of symbols that are at the same level of precedence; so that  $+$  cannot be at the same level of itself. What was a seemingly irrelevant theoretical limit hid a conceptual inadequacy which had had no practical impact until OPL properties were exploited to build parallel compilers.<sup>2</sup> We felt therefore “compelled” to finally remove this relatively disturbing hypothesis: this is the object of the present paper.

As often claimed in previous literature, when discussing the impact of forbidding cycles of operators all at the same level of precedence, removing such a restriction requires allowing grammar *right hand sides (rhs) to include the Kleene-\* operator* (or the  $+$  operator which excludes the empty string from its result). Thus, we introduce the *Cyclic operator-precedence grammars (C-OPGs)* which include the above feature: whereas such a feature is often used in general

---

<sup>1</sup>Of course, assuming that the use of parentheses does not alter the normal precedence between the various operators.

<sup>2</sup>It must also be recalled that in programming languages very long arithmetic expressions are quite unusual; not so in data description languages such JSON.

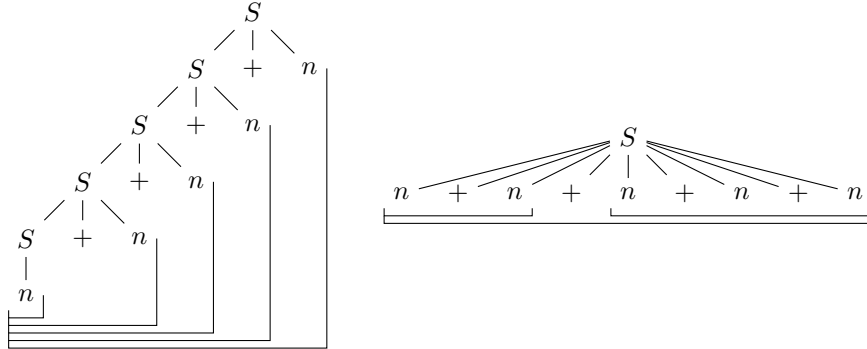


Figure 1: Left-associative syntax tree (left) vs equal-level one (right) of the plus operator. The left syntax tree imposes a sequential left-to-right parsing and semantic processing whereas the right one can be split onto several branches to be partially processed independently and further aggregated.

context-free grammars to make the syntax of programming languages more compact without increasing their expressive power, we show that C-OPGs (unlike OPGs) are fully equivalent to OPAs and other formalisms to define OPLs, such as the MSO logic. We also show that all results previously obtained under the above hypothesis still hold by using C-OPGs instead of traditional OPGs. Although the goal of this paper is not to develop parallel compilation algorithms rooted in C-OPGs, we show how they naturally overcome the difficulty pointed out by [10] and would allow to revisit their techniques, or to improve the efficiency of our previous parallel parser [8].

This paper is the extended version of an article published in the proceedings of DLT 2024 [11], enhanced with more thorough explanations and examples, and proofs of all presented results. The paper is structured as follows: in Section 2, we summarize existing results and notations on OPLs; in Section 3, we introduce C-OPGs and prove their equivalence with OPAs; in Section 4, we compare the expressiveness of C-OPGs with other OPL formalisms; in Section 5, we analyze the closure properties of C-OPGs; in Section 6, we conclude the paper.

## 2. Background

We assume some familiarity with the classical literature on formal language and automata theory, e.g., [12, 13]. Here, we just list and explain our notations for the basic concepts we use from this theory. We denote the terminal alphabet by  $\Sigma$  and the empty string by  $\varepsilon$ . For a string, or set,  $x$ ,  $|x|$  denotes the length, or the cardinality, of  $x$ . The character  $\#$ , not present in the terminal alphabet, is used as string *delimiter*, and we define the alphabet  $\Sigma_{\#} = \Sigma \cup \{\#\}$ . We use the Kleene star operator  $*$  and the  $+$  operator with their usual *regular expression* semantics. We assume that their scope is always marked by parentheses.

## 2.1. Grammars

**Definition 1 (Grammar and language).** A *context-free (CF) grammar* is a tuple  $G = (\Sigma, V_N, P, S)$  where  $\Sigma$  and  $V_N$ , with  $\Sigma \cap V_N = \emptyset$ , are resp. the terminal and the nonterminal alphabets, the total alphabet is  $V = \Sigma \cup V_N$ ,  $P \subseteq V_N \times V^*$  is the rule (or production) set, and  $S \subseteq V_N$ ,  $S \neq \emptyset$ , is the axiom set. For a generic rule, denoted as  $A \rightarrow \alpha$ , where  $A$  and  $\alpha$  are resp. called the left/right hand sides (lhs / rhs), the following forms are relevant:

axiomatic :  $A \in S$   
terminal :  $\alpha \in \Sigma^+$   
empty :  $\alpha = \varepsilon$   
renaming :  $\alpha \in V_N$   
operator :  $\alpha \notin V^*V_NV_NV^*$ , i.e., at least one terminal is interposed between any two nonterminals occurring in  $\alpha$ .

A grammar is called *backward deterministic* or a BD-grammar (or *invertible*) if  $(B \rightarrow \alpha, C \rightarrow \alpha \in P)$  implies  $B = C$ .

If all rules of a grammar are in operator form, the grammar is called an *operator grammar* or O-grammar.

For brevity we take for granted the usual definition of *derivation* denoted by the symbols  $\xRightarrow{G}$  (immediate derivation),  $\xRightarrow{*}{G}$  (reflexive and transitive closure of  $\xRightarrow{G}$ ),  $\xRightarrow{+}{G}$  (transitive closure of  $\xRightarrow{G}$ ),  $\xRightarrow{m}{G}$  (derivation in  $m$  steps); the subscript  $G$  will be omitted whenever clear from the context. We also take for granted the notion of *syntax tree (ST)*. As usual, the *frontier* of a syntax tree is the ordered left to right sequence of the leaves of the tree.

The *language* defined by a grammar starting from a nonterminal  $A$  is

$$L_G(A) = \left\{ w \mid w \in \Sigma^*, A \xRightarrow{*}{G} w \right\}.$$

We call  $w$  a *sentence* if  $A \in S$ . The union of  $L_G(A)$  for all  $A \in S$  is the language  $L(G)$  defined by  $G$ .

Two grammars defining the same language are *equivalent*. Two grammars generating the same set of syntax trees (up to a renaming of their internal nodes) are *structurally equivalent*.

*Notation.* In the following, *unless otherwise explicitly stated*, lowercase letters at the beginning of the alphabet will denote terminal symbols, lowercase letters at the end of the alphabet will denote strings of terminals, Greek letters at the beginning of the alphabet will denote strings in  $V^*$ . Capital letters will be used for nonterminal symbols.

Any grammar can be effectively transformed into an equivalent BD-grammar, and also into an O-grammar [14, 13] without renaming rules and without empty rules but possibly a single rule whose lhs is an axiom not otherwise occurring in any other production. *From now on, w.l.o.g., we exclusively deal with O-grammars without renaming and empty rules, with the only exception that, if  $\varepsilon$*

is part of the language, there is a unique empty rule whose lhs is an axiom that does not appear in the rhs of any production.

## 2.2. Operator precedence grammars

We define operator precedence grammars (OPGs) following primarily [15]. Intuitively, operator precedence grammars are O-grammars whose parsing is driven by three *precedence relations*, called *equal*, *yield* and *take*, included in  $\Sigma_{\#} \times \Sigma_{\#}$ . They are defined in such a way that two consecutive terminals of a grammar's rhs *ignoring possible nonterminals in between* are in the equal relation, while the two extreme ones—again, whether or not preceded or followed by a nonterminal—are preceded by a yield and followed by a take relation, respectively; in this way a complete rhs of a grammar rule is identified and can be *reduced* to a corresponding lhs by a typical bottom-up parsing algorithm. More precisely, the three relations are defined as follows. Subsequently we show how they can drive the bottom-up parsing of sentences.

**Definition 2 ([1]).** Let  $G = (\Sigma, V_N, P, S)$  be an O-grammar. Let  $a, b$  denote elements in  $\Sigma$ ,  $A, B$  in  $V_N$ ,  $C$  either an element of  $V_N$  or the empty string  $\varepsilon$ , and  $\alpha, \beta$  range over  $V^*$ . The *left and right terminal sets* of nonterminals are, respectively:

$$\mathcal{L}_G(A) = \left\{ a \in \Sigma \mid \exists C : A \xrightarrow{*}_G C a \alpha \right\} \text{ and } \mathcal{R}_G(A) = \left\{ a \in \Sigma \mid \exists C : A \xrightarrow{*}_G \alpha a C \right\}.$$

The *operator precedence (OP) relations* are defined over  $\Sigma_{\#} \times \Sigma_{\#}$  as follows:

- equal in precedence:  $a \doteq b \iff \exists A \rightarrow \alpha a C b \beta \in P$
- takes precedence:  $a \triangleright b \iff \exists A \rightarrow \alpha B b \beta \in P, a \in \mathcal{R}(B)$ ,  
and  $a \triangleright \# \iff \exists B \in S, a \in \mathcal{R}(B)$ ;
- yields precedence:  $a \triangleleft b \iff \exists A \rightarrow \alpha a B \beta \in P, b \in \mathcal{L}(B)$ ,  
and  $\# \triangleleft b \iff \exists B \in S, b \in \mathcal{L}(B)$ .

OP relations can be collected into a  $|\Sigma_{\#}| \times |\Sigma_{\#}|$  array, called the *operator precedence matrix* of the grammar,  $OPM(G)$ : for each pair  $(a, b) \in \Sigma_{\#} \times \Sigma_{\#}$ ,  $OPM_{a,b}(G)$  contains the OP relations holding between  $a$  and  $b$ .

Consider a square matrix:  $M = \{M_{a,b} \subseteq \{\doteq, \triangleleft, \triangleright\} \mid a, b \in \Sigma_{\#}\}$ . By convention, either  $M_{\#, \#} = \emptyset$  or  $M_{\#, \#} = \{\doteq\}$ .

- $M$  is called *conflict-free* iff  $\forall a, b \in \Sigma_{\#}, 0 \leq |M_{a,b}| \leq 1$ .
- A conflict-free matrix is called *total* or *complete* iff  $\forall a, b \in \Sigma_{\#}, |M_{a,b}| = 1$ .
- $M$  is  *$\doteq$ -acyclic* if the transitive closure of the  $\doteq$  relation over  $\Sigma \times \Sigma$  is irreflexive.

We extend the set inclusion relations and the Boolean operations in the obvious cell-by-cell way, to any two matrices having the same terminal alphabet. Two matrices are *compatible* iff their union is conflict-free.

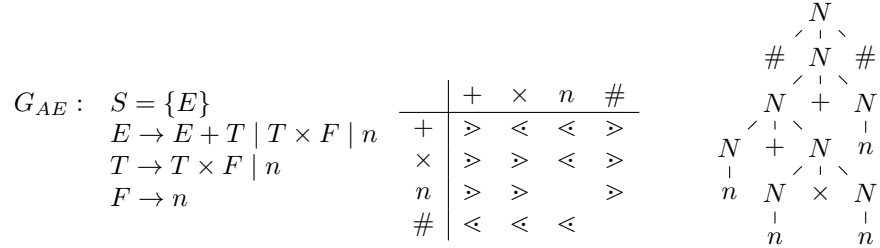


Figure 2:  $G_{AE}$  (left), its OPM (center), and the syntax tree of  $n + n \times n + n$  according to the OPM (right).

**Definition 3 (Operator precedence grammar).** A grammar  $G$  is an *operator precedence grammar* (OPG) iff the matrix  $OPM(G)$  is conflict-free, i.e., the three OP relations are disjoint.

Figure 2 (left) displays an OPG,  $G_{AE}$ , which generates simple, unparenthesized arithmetic expressions and its OPM (center). The left and right terminal sets of  $G_{AE}$ 's nonterminals  $E$ ,  $T$  and  $F$  are, respectively:

$$\begin{aligned} \mathcal{L}(E) &= \{+, \times, n\}, & \mathcal{L}(T) &= \{\times, n\}, & \mathcal{L}(F) &= \{n\}, \\ \mathcal{R}(E) &= \{+, \times, n\}, & \mathcal{R}(T) &= \{\times, n\}, & \mathcal{R}(F) &= \{n\}. \end{aligned}$$

Despite the arithmetic relations having similar typography, the OP relations do not enjoy any of the transitive, symmetric, reflexive properties. We kept the original Floyd's notation but we urge the reader not to be confused by the similarity of the two notations.

It is known that the OPL family is strictly included within the deterministic and reverse-deterministic CF family, i.e., the languages that can be deterministically parsed both from left to right and from right to left [3].

The key feature of OPLs is that a conflict-free OPM  $M$  defines a universe of *strings compatible with  $M$*  and associates to each of them a *unique syntax tree* whose internal nodes are unlabeled and whose leaves are elements of  $\Sigma$ , or, equivalently, a unique parenthesization. We illustrate this feature through a simple example and refer the reader to previous literature for a thorough description of OP parsing [16, 15].

**Example 1.** Consider the  $OPM(G_{AE})$  of Figure 2 and the string  $n + n \times n + n$ . Display all precedence relations holding between consecutive terminal characters, including the relations with the delimiters  $\#$  as shown below:

$$\# < n > + < n > \times < n > + < n > \#$$

each pair  $<, >$  (with no further  $<, >$  in between) includes a *possible* rhs of a production of *any* OPG sharing the OPM with  $G_{AE}$ , not necessarily a  $G_{AE}$  rhs. Thus, as it happens in typical bottom-up parsing, we replace, i.e. *reduce*,

each string included within the pair  $\langle, \rangle$  with a *dummy nonterminal*  $N$ ; this is because nonterminals are irrelevant for OPMs. Notice that the above reductions could be easily performed in *parallel*. The result is the string  $\#N + N \times N + N\#$ . Next, we compute again the precedence relation between consecutive terminal characters by *ignoring nonterminals*: the result is

$$\# \langle N + \langle N \times N \rangle + N \rangle \#$$

This time, there is only one pair  $\langle, \rangle$  including a potential rhs determined by the OPM. Again, we replace the pattern  $N \times N$ , with the dummy nonterminal  $N$ ; notice that there is no doubt about associating the two  $N$  to the  $\times$  rather than to one of the adjacent  $+$  symbols: if we replaced, say, just the  $\times$  with an  $N$  we would obtain the string  $N + NNN + N$  which cannot be derived by an O-grammar. By recomputing the precedence relations we obtain the string  $\# \langle N + N \rangle + N \rangle \#$ . Finally, by replacing twice  $N + N$  with  $N$ , we obtain  $\#N\#$ .

We underline that parsing must always proceed *bottom-up*, by reducing the innermost pair(s) of matched  $\langle$  and  $\rangle$ . Non-innermost pairs do not necessarily match, as in the string  $\#\langle N + N \rangle + N \rangle \#$ , since in this case the  $\langle$  between  $\#$  and the second  $+$  will appear only after reducing  $\langle N + N \rangle$  to  $N$ . The fact that the external  $\langle$  and  $\rangle$  “look matched” in a string such as  $\#\langle n \rangle + \langle n \rangle \times \langle n \rangle + \langle n \rangle \#$  is only coincidental.

The result of the whole bottom-up reduction procedure is synthetically represented by the *syntax tree* of Figure 2 (right), which shows the precedence of multiplication over addition in traditional arithmetic. It also suggests a natural association to the left of both operations: if we reverted the order of the rhs of the rules rewriting  $E$  and  $T$ , the structure of the tree would suggest associativity to the right of both operations. This would not alter the semantics of the two operations, which can indifferently be associated to the left and to the right. Not so is if we dealt with, say, subtraction or division, which instead *impose* association to the left.

Notice that the tree of Figure 2 has been obtained—uniquely and deterministically—by using exclusively the OPM, not the grammar  $G_{AE}$ , although the string  $n + n \times n + n \in L(G_{AE})$ <sup>3</sup>.

Obviously, all sentences of  $L(G_{AE})$  can be given a syntax tree by  $OPM(G_{AE})$ , but there are also strings in  $\Sigma^*$  that can be parsed according to the same OPM but are not in  $L(G_{AE})$ . For example, the string  $+++$  is parsed according to the  $OPM(G_{AE})$  as a ST that associates the  $+$  characters to the left. Notice also that, in general, not every string in  $\Sigma^*$  is assigned a syntax tree by an OPM; e.g., in the case of  $OPM(G_{AE})$  the parsing procedure applied to  $nn$  is immediately blocked since there is no precedence relation between  $n$  and itself.

---

<sup>3</sup>The above procedure that led to the syntax tree of Figure 2 could be easily adapted to become an algorithm that produces a new syntax tree whose internal nodes are labeled by  $G_{AE}$ 's nonterminals. Such an algorithm could be made deterministic by transforming  $G_{AE}$  into an equivalent BD grammar (sharing the same OPM).

The following definition synthesizes the concepts introduced by Example 1.

**Definition 4 (OP-alphabet and Maxlanguage).**

- A string in  $\Sigma^*$  is *compatible* with an OPM  $M$  iff the procedure described in Example 1 terminates by producing the pattern  $\#N\#$ . The set of all strings compatible with an OPM  $M$  is called the *maxlanguage* or the *universe* of  $M$  and is simply denoted as  $L(M)$ .
- Let  $M$  be a conflict-free OPM over  $\Sigma_{\#} \times \Sigma_{\#}$ . We use the same identifier  $M$  to denote the *partial* function  $M$  that assigns to strings in  $\Sigma^*$  their unique ST as informally illustrated in Example 1.
- The pair  $(\Sigma, M)$  where  $M$  is a conflict-free OPM over  $\Sigma_{\#} \times \Sigma_{\#}$ , is called an *OP-alphabet*. We introduce the concept of OP-alphabet as a pair to emphasize that it defines a universe of strings on the alphabet  $\Sigma$ —not necessarily covering the whole  $\Sigma^*$ —and implicitly assigns them a structure univocally determined by the OPM, or, equivalently, by the function  $M$ .
- The class of  $(\Sigma, M)$ -*compatible* OPGs and OPLs are:

$$\mathcal{G}_M = \{G \mid G \text{ is an OPG and } OPM(G) \subseteq M\}, \quad \mathcal{L}_M = \{L(G) \mid G \in \mathcal{G}_M\}.$$

Notice that, if two grammars belonging to the same  $\mathcal{G}_M$  are equivalent, they are also structurally equivalent.

Various formal properties of OPGs and OPLs are documented in the literature, e.g., in [2, 3, 15]. The next proposition recalls those that are relevant for this article. It also shows that *some* of such properties depend critically on restrictions applied to the OPM and formalized in the following definition.

**Definition 5 (Acyclic and rhs-bounded OPGs).**

- An OPG is  $\dot{=}$ -acyclic iff so is its OPM.  $\mathcal{G}_M^{acyc}$  and  $\mathcal{L}_M^{acyc}$  denote, respectively, the class of  $\dot{=}$ -acyclic OPGs and the languages it generates.
- For a positive integer  $q$ ,  $\mathcal{G}_M^q$  denotes the family of OPGs where the production rhs has a length not greater than  $q$ , and  $\mathcal{L}_M^q$  denotes the family of languages it generates.

Notice that any acyclic OPG is in  $\mathcal{G}_M^q$  with  $q = |\Sigma| \cdot 2 + 1$  but obviously the fact that an OPG is in  $\mathcal{G}_M^q$  for any  $q$  does not imply that  $M$  is acyclic.

**Proposition 1 (Algebraic properties of OPGs and OPLs).**

1. If an OPM  $M$  is total, then the corresponding homonymous function is total as well, i.e.,  $L(M) = \Sigma^*$ .
2. Let  $(\Sigma, M)$  be an OP-alphabet. The classes  $\mathcal{G}_M^{acyc}$  and  $\mathcal{G}_M^q$ , for any positive integer  $q$ , contain an OPG, called the *maxgrammar* of  $M$ , denoted by  $G_{max,M}$ , which generates the *maxlanguage*  $L(M)$ . For all grammars  $G$  in the respective classes,  $L(G) \subseteq L(M)$ .

3. The closure properties of the families  $\mathcal{L}_M, \mathcal{L}_M^{acyc}, \mathcal{L}_M^q$ , of  $(\Sigma, M)$ -compatible OPLs defined by a total<sup>4</sup> OPM are the following:

- $\mathcal{L}_M, \mathcal{L}_M^{acyc}, \mathcal{L}_M^q$  are closed under union, intersection and set-difference, therefore also under complement (if a maxgrammar of  $M$  exists).
- $\mathcal{L}_M, \mathcal{L}_M^{acyc}, \mathcal{L}_M^q$  are closed under concatenation.
- $\mathcal{L}_M^{acyc}$  and  $\mathcal{L}_M^q$  are closed under Kleene star.

**On the impact of the hypotheses on the OPM.** From the beginning of the investigation on the algebraic properties of OPLs the  $\dot{=}$ -acyclicity or the rhs-boundedness hypothesis have been assumed to achieve the properties summarized above<sup>5</sup>. Indeed both assumptions do cause some loss of generality in terms of expressive power.

For instance, it is impossible to generate the language  $L_1 = \{a^n(bc)^n\} \cup \{b^n(ca)^n\} \cup \{c^n(ab)^n\}$  by means of an OPG without an OPM that includes the relations  $a \dot{=} b \dot{=} c \dot{=} a$  [4, 17]. As a further consequence, the language  $L_2 = L_1 \cup (abc)^*$  cannot be generated at all by any OPG since the only way to generate strings of the type  $(abc)^n$  for arbitrary  $n$  is to “break” their derivation by means of rules such as  $A \rightarrow abcA$ , but this would affect the original precedence relations. On the other hand it is easy to generate  $L_3 = \{(abc)^n \mid n \leq q\}$  for any fixed  $q$ .

Thus, we have the strict inclusions relations between OPL’s subfamilies:  $\mathcal{L}_M^{acyc} \subsetneq \mathcal{L}_M^q \subsetneq \mathcal{L}_M$ , with  $L_2 \notin \mathcal{L}_M$ . We will also soon see that pushdown automata explicitly designed to accept languages defined on a given OP-alphabet do not suffer from the above limitation.

Nevertheless, despite the loss in terms of expressive power, either one of the two hypotheses on the OPM has been consistently assumed in favor of a simpler mathematical notation and less involved technical proofs. Such a decision has been motivated by the fact that both hypotheses appeared to be of purely mathematical interest, but no practical case in compilation or automatic property verification was found where such hypotheses could not be guaranteed. As we anticipated in the introduction, this is not anymore the case.

As a final remark, since a conflict-free OPM assigns to each string at most one ST—and exactly one if the OPM is complete—the above closure properties of OPLs w.r.t. Boolean operations automatically extend to sets of their STs. The same does not apply to the case of concatenation which in general may produce significant reshaping of the original STs [3]. Furthermore, any complete, conflict-free,  $\dot{=}$ -acyclic or rhs-bounded OPM, defines a *universe of STs* whose frontiers are  $\Sigma^*$ .

---

<sup>4</sup>The assumption for on OPM of being total does not imply any loss of generality since any OPM can be arbitrarily extended to a total one.

<sup>5</sup>Initially the rhs-boundedness was chosen [2], but later on the “almost equivalent”  $\dot{=}$ -acyclicity has been preferred in favor of notational simplification.

### 2.3. Chains and Operator Precedence Automata (OPA)

The notion of *chain* introduced next is an alternative way to represent STs where internal nodes are irrelevant and “anonymized”.

**Definition 6 (Chains).** Let  $(\Sigma, M)$  be an OP-alphabet.

- A *simple chain* is a word  $a_0a_1a_2 \dots a_na_{n+1}$ , written as  ${}^{a_0}[a_1a_2 \dots a_n]^{a_{n+1}}$ , such that:  $a_0, a_{n+1} \in \Sigma \cup \{\#\}$ ,  $a_i \in \Sigma$  for every  $i : 1 \leq i \leq n$ ,  $M_{a_0a_{n+1}} \neq \emptyset$ , and  $a_0 < a_1 \doteq a_2 \dots a_{n-1} \doteq a_n > a_{n+1}$ .
- A *composed chain* is a word  $a_0x_0a_1x_1a_2 \dots a_nx_na_{n+1}$ , with  $x_i \in \Sigma^*$ , where  ${}^{a_0}[a_1a_2 \dots a_n]^{a_{n+1}}$  is a simple chain, and either  $x_i = \varepsilon$  or  ${}^{a_i}[x_i]^{a_{i+1}}$  is a chain (simple or composed), for every  $i : 0 \leq i \leq n$ . Such a composed chain will be written as  ${}^{a_0}[x_0a_1x_1a_2 \dots a_nx_n]^{a_{n+1}}$ .
- The *body* of a chain  ${}^a[x]^b$ , simple or composed, is the word  $x$ .
- Given a chain  ${}^a[x]^b$  the *depth*  $d(x)$  of its body  $x$  is defined recursively:  $d(x) = 1$  if the chain is simple, whereas  $d(x_0a_1x_1 \dots a_nx_n) = 1 + \max_i d(x_i)$ . The depth of a chain is the depth of its body.

For instance, the ST of Figure 2 (right) is biunivocally represented by the composed chain  $\#[x_0 + x_1]\#$ , where, in turn  $x_0$  is the body of the composed chain  $\#[y_0 + y_1]^+$ ,  $y_0$  is the body of the simple chain  $\#[e]^+$ ,  $y_1$  is the body of the composed chain  $^+[z_0 \times z_1]^+$ , etc. The depth of the main chain is 5.

As well as an OPG selects a set of STs within the universe defined by its OPM, an *operator precedence automaton (OPA)* selects a set of chains within the universe defined by an OP-alphabet.

**Definition 7 (Operator precedence automaton).** A nondeterministic *operator precedence automaton (OPA)* is a tuple:  $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$  where:

- $(\Sigma, M)$  is an operator precedence alphabet,
- $Q$  is a set of states (disjoint from  $\Sigma$ ),
- $I \subseteq Q$  is a set of *initial* states,
- $F \subseteq Q$  is a set of *final* states,
- $\delta$ , named *transition function*, is a triple of functions:

$$\delta_{\text{shift}} : Q \times \Sigma \rightarrow \wp(Q) \quad \delta_{\text{push}} : Q \times \Sigma \rightarrow \wp(Q) \quad \delta_{\text{pop}} : Q \times Q \rightarrow \wp(Q)$$

We represent a nondeterministic OPA by a graph with  $Q$  as the set of vertices and  $\Sigma \cup Q$  as the set of edge labelings. The edges of the graph are denoted by different shapes of arrows to distinguish the three types of transitions: there is an edge  $q \xrightarrow{a} p$  (respectively,  $q \xrightarrow{-a} p$ ) from state  $q$  to state  $p$  labeled by  $a \in \Sigma$  denoted by a normal (respectively, dashed) arrow if and only if  $p \in \delta_{\text{push}}(q, a)$

(respectively,  $p \in \delta_{\text{shift}}(q, a)$ ) and there is an edge  $q \xrightarrow{r} p$  from state  $q$  to state  $p$  labeled by  $r \in Q$  and denoted by a double arrow if and only if  $p \in \delta_{\text{pop}}(q, r)$ .

To define the semantics of the automaton, we introduce some notations.

We use letters  $p, q, p_i, q_i, \dots$  to denote states in  $Q$ . Let  $\Gamma$  be  $\Sigma \times Q$  and let  $\Gamma'$  be  $\Gamma \cup \{\perp\}$ ; we denote symbols in  $\Gamma'$  as  $[a, q]$  or  $\perp$ . We set  $\text{symbol}([a, q]) = a$ ,  $\text{symbol}(\perp) = \#$ , and  $\text{state}([a, q]) = q$ . Given a string  $\Pi = \perp \pi_1 \pi_2 \dots \pi_n$ , with  $\pi_i \in \Gamma$ ,  $n \geq 0$ , we set  $\text{symbol}(\Pi) = \text{symbol}(\pi_n)$ , including the particular case  $\text{symbol}(\perp) = \#$ .

A *configuration* of an OPA is a triple  $C = \langle \Pi, q, w \rangle$ , where  $\Pi \in \perp \Gamma^*$ ,  $q \in Q$  and  $w \in \Sigma^* \#$ . The first component represents the contents of the stack, the second component represents the current state of the automaton, while the third component is the part of input still to be read.

A *computation* or *run* of the automaton is a finite sequence of *moves* or *transitions*  $C_1 \vdash C_2$ ; there are three kinds of moves, depending on the precedence relation between the symbol on top of the stack and the next symbol to read:

**push move:** if  $\text{symbol}(\Pi) \leq a$  then  $\langle \Pi, p, ax \rangle \vdash \langle \Pi[a, p], q, x \rangle$ , with  $q \in \delta_{\text{push}}(p, a)$ ;

**shift move:** if  $a \doteq b$  then  $\langle \Pi[a, p], q, bx \rangle \vdash \langle \Pi[b, p], r, x \rangle$ , with  $r \in \delta_{\text{shift}}(q, b)$ ;

**pop move:** if  $a \succ b$  then  $\langle \Pi[a, p], q, bx \rangle \vdash \langle \Pi, r, bx \rangle$ , with  $r \in \delta_{\text{pop}}(q, p)$ .

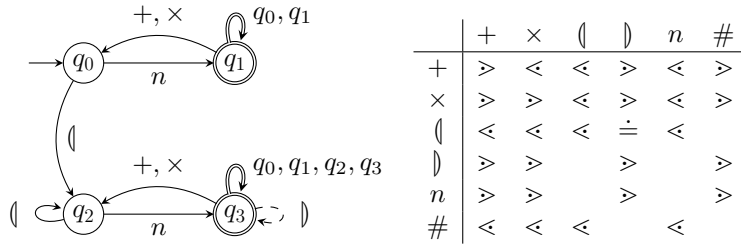
Shift and pop moves are never performed when the stack contains only  $\perp$ . Push and shift moves update the current state of the automaton according to the transition functions  $\delta_{\text{push}}$  and  $\delta_{\text{shift}}$ , respectively: push moves put a new element on the top of the stack consisting of the input symbol together with the current state of the automaton, whereas shift moves update the top element of the stack by changing its input symbol only. The pop move removes the symbol on the top of the stack, and the state of the automaton is updated by  $\delta_{\text{pop}}$  on the basis of the pair of states consisting of the current state of the automaton and the state of the removed stack symbol; notice that in this move the input symbol is used only to establish the  $\succ$  relation and it remains available for the following move.

A configuration  $\langle \perp, q_I, x\# \rangle$  is *initial* if  $q_I \in I$ ; a configuration  $\langle \perp, q_F, \# \rangle$  is *accepting* if  $q_F \in F$ . The language accepted by the automaton is:

$$L(\mathcal{A}) = \left\{ x \mid \langle \perp, q_I, x\# \rangle \vdash^* \langle \perp, q_F, \# \rangle, q_I \in I, q_F \in F \right\}.$$

Any nondeterministic OPA can be transformed into a deterministic one at the cost of quadratic exponential increase in the size of the state space [4].

**Example 2.** The OPA depicted in Figure 3 (top, left) based on the OPM that the (top, right) accepts the language of arithmetic expressions enriched w.r.t  $L(G_{AE})$  in that it introduces the use of explicit parentheses to alter the natural precedence of arithmetic operations. The same figure (bottom) also shows an accepting computation on input  $n + n \times (n + n)$ .



stack	state	current input
$\perp$	$q_0$	$n + n \times (n + n)\#$
$\perp[n, q_0]$	$q_1$	$+n \times (n + n)\#$
$\perp$	$q_1$	$+n \times (n + n)\#$
$\perp[+, q_1]$	$q_0$	$n \times (n + n)\#$
$\perp[+, q_1][n, q_0]$	$q_1$	$\times(n + n)\#$
$\perp[+, q_1]$	$q_1$	$\times(n + n)\#$
$\perp[+, q_1][\times, q_1]$	$q_0$	$(n + n)\#$
$\perp[+, q_1][\times, q_1][(, q_0]$	$q_2$	$n + n)\#$
$\perp[+, q_1][\times, q_1][(, q_0)[n, q_2]$	$q_3$	$+n)\#$
$\perp[+, q_1][\times, q_1][(, q_0]$	$q_3$	$+n)\#$
$\perp[+, q_1][\times, q_1][(, q_0)[+, q_3]$	$q_2$	$n)\#$
$\perp[+, q_1][\times, q_1][(, q_0)[+, q_3][n, q_2]$	$q_3$	$)\#$
$\perp[+, q_1][\times, q_1][(, q_0)[+, q_3]$	$q_3$	$)\#$
$\perp[+, q_1][\times, q_1][(, q_0]$	$q_3$	$)\#$
$\perp[+, q_1][\times, q_1][), q_0]$	$q_3$	$\#$
$\perp[+, q_1][\times, q_1]$	$q_3$	$\#$
$\perp[+, q_1]$	$q_3$	$\#$
$\perp$	$q_3$	$\#$

Figure 3: An OPA (top, left), its OPM (top, right) and an example of computation for the language of Example 2 (bottom). Arrows  $\rightarrow$ ,  $\dashrightarrow$  and  $\Longrightarrow$  denote push, shift and pop transitions, respectively. To avoid confusion with the overloaded parenthesis symbols, the parentheses used as terminal symbols are denoted as  $($  and  $)$ .

Notice, also, how easy it is to build an OPA recognizing the language  $L_2$  defined in Subsection 2.1 with its necessarily cyclic OPM. This remark argues in favor of greater expressive power of OPAs over traditional OPGs, whether restricted or not, as it will be completely proved in the following sections.

**Definition 8.** Let  $\mathcal{A}$  be an OPA. A *support* for a simple chain  $a_0[a_1a_2\dots a_n]^{a_{n+1}}$  is any path in  $\mathcal{A}$  of the form

$$q_0 \xrightarrow{a_1} q_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \dashrightarrow^{a_n} q_n \xRightarrow{q_0} q_{n+1} \quad (1)$$

Notice that the label of the last (and only) pop is exactly  $q_0$ , i.e. the first state of the path; this pop is executed because of relations  $a_0 < a_1$  and  $a_n > a_{n+1}$ . A *support for the composed chain*  $a_0[x_0a_1x_1a_2\dots a_nx_n]^{a_{n+1}}$  is any path in  $\mathcal{A}$  of the form

$$q_0 \xrightarrow{x_0} q'_0 \xrightarrow{a_1} q_1 \xrightarrow{x_1} q'_1 \dashrightarrow^{a_2} \dots \dashrightarrow^{a_n} q_n \xrightarrow{x_n} q'_n \xRightarrow{q'_0} q_{n+1} \quad (2)$$

where for every  $i : 0 \leq i \leq n$ :

- if  $x_i \neq \varepsilon$ , then  $q_i \xrightarrow{x_i} q'_i$  is a support for the (simple or composed) chain  $a_i[x_i]^{a_{i+1}}$
- if  $x_i = \varepsilon$ , then  $q'_i = q_i$ .

Notice that the label of the last pop is exactly  $q'_0$ .

The support of a chain with body  $x$  will be denoted by  $q_0 \xrightarrow{x} q_{n+1}$ .

The context  $a, b$  of a chain  $a[x]b$  is used by the automaton to build its support only because  $a < x$  and  $x > b$ ; thus, the chain's body contains all information needed by the automaton to build the subtree whose frontier is that string, once it is understood that its first move is a push and its last one is pop. This is a distinguishing feature of OPLs, not shared by other deterministic languages: we call it the *locality principle* of OPLs, which has been exploited to build parallel and/or incremental OP parsers [7].

### 3. Cyclic Operator Precedence Grammars (C-OPGs) and their equivalence with OPAs

Proposition 1 shows that *some, but not all*, of the algebraic properties of OPLs depend critically on the  $\doteq$ -acyclicity hypothesis. This is due to the fact that without such a hypothesis the rhs of an OPG have an unbounded length but cannot be infinite: e.g., no OPG can generate the language  $\{a, b\}^*$  if  $a \doteq b$  and  $b \doteq a$ . In most cases cycles of this type can be “broken” as it has been done up to now, e.g., to avoid the  $\doteq$  relation in arithmetic expressions by associating the operator indifferently to the right or to left. Thus, although it is known that, from a theoretical point of view, the  $\doteq$ -acyclicity hypothesis affects the expressive power of OPGs, we accepted so far the  $\doteq$ -acyclicity hypothesis to keep the notation as simple as possible. Recently, however, Li and Taura [10] observed that such a restriction may hamper the benefits achievable by the

parallel compilation techniques that exploit the local parsability property of OPLs [8]. Thus, it is time to introduce the necessary extension of OPGs so that the  $\doteq$ -acyclicity hypothesis can be avoided and they become fully equivalent to other formalisms to define OPLs.

**Definition 9 (Cyclic Operator Precedence Grammar).**

- A  $^+$ -O-expression on  $V^*$  is an expression obtained from the elements of  $V$  by iterative application of concatenation and the usual  $^+$  operator<sup>6</sup>, provided that any substring thereof has no two adjacent nonterminals; for convenience, and w.l.o.g., we assume that all subexpressions that are argument of the  $^+$  operator are terminated by a terminal character.
- A Cyclic O-grammar (C-OG) is an O-grammar whose production rhs are  $^+$ -O-expressions.
- For a grammar rule  $A \rightarrow \alpha$  of an C-OG, the  $\xRightarrow[G]{}$  (immediate derivation) relation is defined as  $\beta A \gamma \xRightarrow[G]{\alpha} \beta \zeta \gamma$  iff  $\zeta$  is a string belonging to the language defined by the  $^+$ -O-expression  $\alpha$ ,  $L(\alpha)$ .
- The equal in precedence relation is redefined as  $a \doteq b$  iff  $\exists A \rightarrow \alpha \wedge \exists \zeta = \eta a B b \theta \mid (B \in V_N \cup \{\varepsilon\} \wedge \zeta \in L(\alpha))$ . The other precedence relations remain defined as for non-cyclic O-grammars.
- A C-OG is a cyclic operator precedence grammar (C-OPG) iff its OPM is conflict-free.

As a consequence of the definition of the immediate derivation relation for C-OPGs the syntax-trees derived therefrom can be unranked, i.e., their internal nodes may have an unbounded number of children.

**Example 3.** The C-OPG depicted in Figure 4 with its OPM generates a fairly complete language of—not necessarily fully—parenthesized arithmetic expressions involving the four basic operations: as usual the multiplicative operations take precedence over the additive ones; furthermore subtraction takes precedence over sum and division over multiplication. The key novelty w.r.t. the traditional way of formalizing arithmetic expressions by means of OPGs are the  $+ \doteq +$  and  $\times \doteq \times$  OP relations; on the contrary we kept the structure that associates subtraction and division to the left, so that the grammar’s STs now fully reflect the semantics of the arithmetic operations. An example ST generated by this grammar is given in Figure 5.

Notice that  $G_{A-AE}$  is purposely ambiguous to keep it compact. To support deterministic parsing it should be transformed into a BD—and therefore unambiguous—form.

---

<sup>6</sup>For our purposes  $^+$  is more convenient than  $^*$  without affecting the generality.

$G_{A-AE}$  :

$S = \{P\}$   
 $P \rightarrow (M+)^+ M \mid M - T \mid (D \times)^+ D \mid D/F \mid (P) \mid n$   
 $M \rightarrow M - T \mid (D \times)^+ D \mid D/F \mid (P) \mid n$   
 $T \rightarrow (D \times)^+ D \mid D/F \mid (P) \mid n$   
 $D \rightarrow D/F \mid (P) \mid n$   
 $F \rightarrow (P) \mid n$

	+	-	×	/	(	)	n	#
+	≐	<	<	<	<	>	<	>
-	>	>	<	<	<	>	<	>
×	>	>	≐	<	<	>	<	>
/	>	>	>	>	<	>	<	>
(	<	<	<	<	<	≐	<	
)	>	>	>	>		>		>
n	>	>	>	>		>		>
#	<	<	<	<	<		<	

Figure 4: A C-OPG (top) and its OPM (bottom).

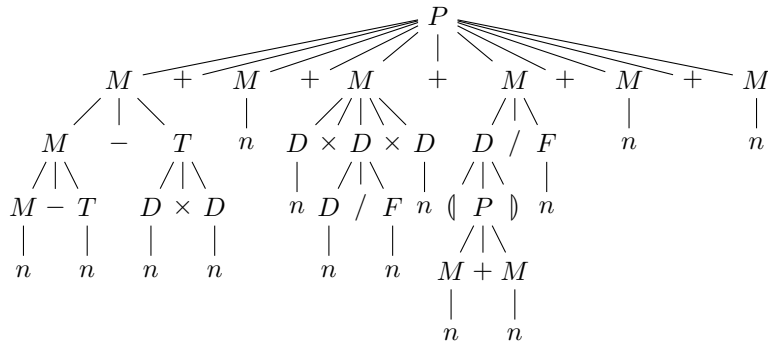


Figure 5: A ST generated by the C-OPG of Figure 4.

By looking at the ST of Figure 5 and comparing it with the original Figure 1, one can get a first intuition of why the introduction of cyclic  $\doteq$  can support more effective parallel compilation algorithms for OPLs. Parallel parsing and compilation for OPLs is rooted in the *local parsability property* of this family: thanks to this property any fragment of input string enclosed within a pair of corresponding  $\langle$  and  $\rangle$  OP relations can be processed in parallel with other similar fragments. However, if, say, the  $+$  operator is associated to the left (or right) as in the case of the left ST of Figure 1 and that of Figure the parsing of a sequence of  $+$  must necessarily proceed sequentially from left to right. Conversely, if the ST has a structure like that of the right tree of Figure 1 and that of Figure 5 the sequence of  $+$ , whether intermixed or not with other subtrees, can be arbitrarily split into several branches which can be parsed and compiled in parallel and, after that, can be joined into a unique subtree as imposed by the  $\doteq$  OP relation between the corresponding extreme terminals of contiguous branches.

### 3.1. Equivalence between C-OPGs and OPAs

The equivalence is obtained by a modification of the analogous proof given in [4] where the additional hypothesis of  $M$  being  $\doteq$ -acyclic was exploited.

First, we describe a procedure to build an OPA equivalent to a C-OPG. Then, we provide the converse construction.

**Theorem 2 (From C-OPGs to OPAs).** *Let  $(\Sigma, M)$  be an OP-alphabet. For any C-OPG defined thereon an equivalent OPA can be effectively built.*<sup>7</sup>

PROOF. A nondeterministic OPA  $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$  equivalent to given C-OPG  $G$  with the same precedence matrix  $M$  as  $G$  is built in such a way that a successful computation thereof corresponds to building bottom-up a syntax tree of  $G$ : the automaton performs a push transition when it reads the first terminal of a new rhs and a shift transition when it reads a terminal symbol inside a rhs, i.e. a leaf with some left sibling leaf. It performs a pop transition when it completes the recognition of a rhs, then it guesses (nondeterministically) the nonterminal at the lhs. Each state contains two pieces of information: the first component represents the prefix of the rhs under construction, whereas the second component is used to recover the rhs *previously under construction* (see Figure 6) whenever all rhs nested below have been completed. Precisely, the construction of  $\mathcal{A}$  is defined as follows.

Let  $\hat{P}$  be the set of rhs  $\gamma$  where all  $+$  and related parentheses have been erased. Let  $\tilde{P}$  be the set of strings  $\tilde{\gamma} \in V^+$  belonging to the language of some rhs  $\gamma$  of  $P$  that is inductively defined as follows:

- a. If  $(\eta)^+$  is a subexpression of  $\gamma$  such that  $\eta$  is a single string  $\in V^+$  then  $\tilde{\eta} = \{\eta, \eta\eta\}$ ;

---

<sup>7</sup>By *effectively*, we mean that the proof outlines a construction that allows one to obtain an OPA from a C-OPG in practice.

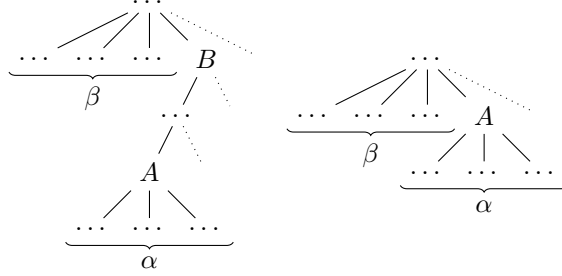


Figure 6: When parsing  $\alpha$ , the prefix previously under construction is  $\beta$ .

- in the particular case that  $P$  contains several productions of the form  $A_i \rightarrow \alpha\eta^{k_i}\theta_i$  or of the form  $A_i \rightarrow \alpha(\eta^{k_i})^+\theta_i$  with  $k_i > 0$ , we set  $\bar{k}$  as the least common multiple of the set  $\{k_i \in \mathbb{N} \mid A_i \rightarrow \alpha\eta^{k_i}\theta_i \text{ or } A_i \rightarrow \alpha(\eta^{k_i})^+\theta_i \in P\}$ , and define  $\tilde{\eta} = \{\eta^{\bar{k}}, \eta^{2\bar{k}}\}$ .
- b. If  $(\eta)^+$  is a subexpression of  $\gamma$  such that  $\eta = \alpha_1(\beta_1)^+\alpha_2(\beta_2)^+ \dots \alpha_n$  where  $\alpha_i \in V^*$ , then  $\tilde{\eta} = \{\eta_1, \eta_1\eta_1\}$  where  $\eta_1 = \alpha_1\tilde{\beta}_1\alpha_2\tilde{\beta}_2 \dots \alpha_n$ ;
- the particular case where  $P$  contains several productions with a common prefix and some of the  $\beta_i$  are of the type  $\lambda_i^{k_i}$  is treated in the same way as in point a. above.

For instance, let  $\eta$  be  $(Ba(bc)^+)^+$ : then  $\hat{\eta} = \{Babc\}$  and  $\tilde{\eta} = \{Babc, Babcbc, Babcbabc, Babcbcbabc, Babcbcbcbabc\}$ .

Let  $\mathbb{P} = \{\alpha \in V^*\Sigma \mid \exists A \rightarrow \eta \in P \wedge \exists \alpha, \beta \mid \alpha\beta \in \tilde{\eta}\}$  be the set of prefixes, ending with a terminal symbol, of strings  $\in \tilde{P}$ ; define  $\mathbb{Q} = \{\varepsilon\} \cup \mathbb{P} \cup V_N$ ,  $Q = \mathbb{Q} \times (\{\varepsilon\} \cup \mathbb{P})$ ,  $I = \{(\varepsilon, \varepsilon)\}$ , and  $F = S \times \{\varepsilon\} \cup \{(\varepsilon, \varepsilon) \mid \varepsilon \in L(G)\}$ . Note that  $|\mathbb{Q}| = 1 + |\mathbb{P}| + |V_N|$  is  $O(m^h)$  where  $m$  is the maximum length of the rhs in  $P$ , and  $h$  is the maximum nesting level of  $^+$  operators in rhs; therefore  $|\mathbb{Q}|$  is  $O(m^{2h})$ .

The transition functions are defined by the following formulas, for  $a \in \Sigma$  and  $\alpha, \alpha_1, \alpha_2 \in \mathbb{Q}$ ,  $\beta, \beta_1, \beta_2 \in \{\varepsilon\} \cup \mathbb{P}$ , and where for any expression  $\xi$ , string  $\bar{\xi}$  is obtained from  $\xi$  by erasing parentheses and  $^+$  operators:

$$\begin{aligned}
 \bullet \delta_{\text{shift}}(\langle \alpha, \beta \rangle, a) \ni & \begin{cases} \text{if } \alpha \notin V_N: & \begin{cases} \langle \bar{\eta}\bar{\zeta}, \beta \rangle & \text{if } \begin{pmatrix} \exists A \rightarrow \gamma \mid \gamma = \eta(\zeta)^+\theta \wedge \\ \alpha a = \bar{\eta}\bar{\zeta}\bar{\zeta} \wedge \\ \alpha a \bar{\theta} \in L(\gamma) \cap \tilde{P} \end{pmatrix} \\ \langle \alpha a, \beta \rangle & \text{otherwise} \end{cases} \\ \text{if } \alpha \in V_N: & \begin{cases} \langle \bar{\eta}\bar{\zeta}, \beta \rangle & \text{if } \begin{pmatrix} \exists A \rightarrow \gamma \mid \gamma = \eta(\zeta)^+\theta \wedge \\ \beta \alpha a = \bar{\eta}\bar{\zeta}\bar{\zeta} \wedge \\ \beta \alpha a \bar{\theta} \in L(\gamma) \cap \tilde{P} \end{pmatrix} \\ \langle \beta \alpha a, \beta \rangle & \text{otherwise} \end{cases} \end{cases} \\
 \bullet \delta_{\text{push}}(\langle \alpha, \beta \rangle, a) \ni & \begin{cases} \langle a, \alpha \rangle & \text{if } \alpha \notin V_N \\ \langle \alpha a, \beta \rangle & \text{if } \alpha \in V_N \end{cases}
 \end{aligned}$$

- $\delta_{\text{pop}}(\langle \alpha_1, \beta_1 \rangle, \langle \alpha_2, \beta_2 \rangle) \ni \langle A, \gamma \rangle$   
for every  $A$  such that  $\begin{cases} \text{if } \alpha_1 \notin V_N : A \rightarrow \alpha \in P \wedge \alpha_1 \in L(\alpha) \cap \hat{P} \\ \text{if } \alpha_1 \in V_N : A \rightarrow \alpha \in P \wedge \beta_1 \alpha_1 \in L(\alpha) \cap \hat{P} \end{cases}$   
and  $\gamma = \begin{cases} \alpha_2 & \text{if } \alpha_2 \notin V_N \\ \beta_2 & \text{if } \alpha_2 \in V_N. \end{cases}$

The states reached by push and shift transitions have the first component in  $\mathbb{P}$ . If state  $\langle \alpha, \beta \rangle$  is reached after a push transition, then  $\alpha$  is the prefix of the rhs (deprived of the  $^+$  operators) that is currently under construction and  $\beta$  is the prefix previously under construction; in this case  $\alpha$  is either a terminal symbol or a nonterminal followed by a terminal one.

If the state is reached after a shift transition, and the  $\alpha$  component of the previous state was not a single nonterminal, then the new  $\alpha$  is the concatenation of the first component of the previous state with the read character. If, instead, the  $\alpha$  component of the previous state was a single nonterminal—which was produced by a pop transition—then the new  $\alpha$  also includes the previous  $\beta$  (see Figure 6) and  $\beta$  is not changed from the previous state. However, if the new  $\alpha$  becomes such that a suffix thereof is a double occurrence of a string  $\zeta \in L((\zeta)^+)$ —hence  $\alpha \in \mathbb{P}$ —then the second occurrence of  $\zeta$  is cut from the new  $\alpha$ , which therefore becomes a prefix of an element of  $\hat{P}$ .

The states reached by a pop transition have the first component in  $V_N$ : if  $\langle A, \gamma \rangle$  is such a state, then  $A$  is the corresponding lhs, and  $\gamma$  is the prefix previously under construction.

For instance, imagine that a C-OPG contains the rules  $A \rightarrow (Ba(bc)^+)^+a$  and  $B \rightarrow h$  and that the corresponding OPA  $\mathcal{A}$  parses the string  $habcbchabca$ : after scanning the prefix  $habcb$   $\mathcal{A}$  has reduced  $h$  to  $B$  and has  $Babcb$  as the first component of its state; after reading the new  $c$  it recognizes that the suffix of the first state component would become a second instance of  $bc$  belonging to  $(bc)^+$ ; thus, it goes back to  $Babcb$ . Then, it proceeds with a new reduction of  $h$  to  $B$  and, when reading with a shift the second  $a$  appends  $Ba$  to its current  $\beta$  which was produced by the previous pop so that the new  $\alpha$  becomes  $BabcBa$ ; after shifting  $b$  it reads  $c$  and realizes that its new  $\alpha$  would become  $BabcBabc$ , i.e., an element of  $(Ba(bc)^+)^+$  and therefore “cuts” it to the single instance thereof, i.e.,  $Babc$ . Finally, after having shifted the last  $a$  it is ready for the last pop.

Notice that the result of  $\delta_{\text{shift}}$  and  $\delta_{\text{push}}$  is a singleton, whereas  $\delta_{\text{pop}}$  may produce several states, in case of repeated rhs. Thus, if  $G$  is BD, the corresponding  $\mathcal{A}$  is deterministic.

The equivalence between  $G$  and  $\mathcal{A}$  derives from the following Lemmas 3 and 4, when  $\beta = \gamma = \varepsilon$ ,  $\Pi = \perp$  and  $A$  is an axiom.  $\square$

The statements of Lemmas 3 and 4 are identical to those of Lemmas 3.2 and 3.3 given in [4] for the original construction applied to non-C-OPGs. The proofs are, however, different, because they have to take  $^+$  operators into account. The key difference is that during a series of shift transitions scanning a simple chain, the  $\beta$  component of the state remains unchanged, whereas the  $\alpha$  component at each transition appends the read character—always under the constraint that

it belongs to  $\mathbb{P}$ —up to the point where the last append would repeat a suffix of  $\alpha$ —say a string  $y$  in case of simple chains—identically; if  $y$  occurs under the  $^+$  operator of a rhs of which  $\alpha$  is a prefix, then the second occurrence of  $y$  is erased and the state is the same as after reading its first occurrence, thus closing a loop in the same way as it happens with finite state machines, because the stack too is identical to the stack after the first scanning of  $y$ . As a consequence, the OPA can repeat the loop reading  $y$  any number of times just as the C-OPG can generate, in an immediate derivation, any number of occurrences of  $y$  thanks to the  $^+$  operator.

**Lemma 3.** *Let  $x$  be the body of a chain  ${}^a[x]^b$ , and  $\beta, \gamma \in \mathbb{P} \cup \{\varepsilon\}$ . Then  $\langle \beta, \gamma \rangle \overset{x}{\rightsquigarrow} q$  implies the existence of  $A \in V_N$  such that  $A \overset{*}{\Rightarrow} x$  in  $G$  and  $q = \langle A, \beta \rangle$ .*

PROOF. We proceed by induction on the chain nesting depth  $d$  of  ${}^a[x]^b$ . When  $d = 1$ ,  ${}^a[x]^b$  is a simple chain. The OPA reads it with a support of the form (1), where  $x = a_1 \dots a_n$ ,  $q_0 = \langle \beta, \gamma \rangle$ , and  $q_{n+1} = q$ . By the definition of the push and shift transition functions, and because by hypothesis  $\beta \notin V_N$ , we have  $q_i = \langle a'_1 \dots a'_{j_i}, \beta \rangle$  for all  $i = 1, \dots, n$ , such that  $a'_1 \dots a'_{j_i}$ , with  $j_i \leq i$ , is the string obtained from  $a_1 \dots a_i$  by erasing all instances occurring consecutively more than once of substrings  $y$  that appear under the  $^+$  operator in the rhs of some production in  $G$ . In particular, for each sequence  $a_h \dots a_{h+k-1} a_{h+k} \dots a_{h+2k-1}$  such that  $a_{h+k} \dots a_{h+2k-1} = a_h \dots a_{h+k-1}$ <sup>8</sup> we have  $q_{h+k}, \dots, q_{h+2k-1} = q_h, \dots, q_{h+k-1}$  by construction of  $\delta_{\text{shift}}$ . By the definition of  $\delta_{\text{pop}}$  and because  $\beta \notin V_N$ , there exists some production  $A \rightarrow \alpha$  such that  $x \in L(\alpha)$  and  $q = \langle A, \beta \rangle$ . Therefore  $A \overset{*}{\Rightarrow} x$ .

For  $d > 1$ ,  ${}^a[x]^b$  is a composed chain and we write its body as  $x = x_0 a_1 x_1 \dots a_n x_n$  and its support as in (2) with  $q_i = \langle \beta_i, \gamma_i \rangle$  for each  $i = 0, \dots, n+1$  and, in particular,  $\langle \beta_0, \gamma_0 \rangle = \langle \beta, \gamma \rangle$  and  $q_{n+1} = q$ . Each  $x_i \neq \varepsilon$  is the body of a chain of depth lower than  $d$ , hence by the inductive hypothesis there exists  $X_i \in V_N$  such that  $X_i \overset{*}{\Rightarrow} x_i$  and  $q'_i = \langle X_i, \beta_i \rangle$ . The support has thus the form

$$\langle \beta, \gamma \rangle \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} \langle \beta_1, \gamma_1 \rangle \overset{x_1}{\rightsquigarrow} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} \langle \beta_n, \gamma_n \rangle \overset{x_n}{\rightsquigarrow} q'_n \xrightarrow{q'_0} q$$

where, for each  $i$ , either  $q'_i = \langle \beta_i, \gamma_i \rangle$  if  $x_i = \varepsilon$  or  $q'_i = \langle X_i, \beta_i \rangle$  otherwise. By the definitions of  $\delta_{\text{push}}$  and  $\delta_{\text{shift}}$ , we can write  $\beta_i = X'_0 a'_1 X'_1 \dots X'_{j_i}$  with  $j_i \leq i$ , and each  $X'_k = \varepsilon$  if  $x_k = \varepsilon$  and  $X_k \in V_N$  otherwise. Again, we obtain  $X'_0 a'_1 X'_1 \dots X'_{j_i}$  by erasing from  $X_0 a_1 X_1 \dots X_i$  all substrings appearing within a  $^+$  operator so that they do not occur more than once. For each sequence  $a_h X_h \dots X_{h+k-1} a_{h+k} X_{h+k} \dots X_{h+2k-1}$  such that  $a_h X_h \dots X_{h+k-1} = a_{h+k} X_{h+k} \dots X_{h+2k-1}$  we have  $q_{h+k}, \dots, q'_{h+2k-1} = q_h, \dots, q'_{h+k-1}$ . State  $q$  then depends on whether  $x_0$  and  $x_n$  are empty, leading exactly to the four cases considered by the definition of  $\delta_{\text{pop}}$ . Eventually, we have  $q = \langle A, \beta \rangle$  for some  $A \in V_N$  such that  $G$  has a production  $A \rightarrow \alpha$  and  $x \in L(\alpha)$ .  $\square$

<sup>8</sup>In point **a.** of the construction in Theorem 2,  $k$  is a multiple of the length  $|y|$  of  $y$ . This is uncommon in practice, since  $k$  is usually equal to  $|y|$ .

**Lemma 4.** *Let  $x$  be the body of a chain and  $A \in V_N$ . Then,  $A \xrightarrow{*} x$  in  $G$  implies  $\langle \beta, \gamma \rangle \xrightarrow{x} \langle A, \beta \rangle$  for every  $\beta, \gamma \in \mathbb{P} \cup \{\varepsilon\}$ .*

PROOF. We carry out the proof by induction on the chain nesting depth  $d$  of  $x$ .

For  $d = 1$ ,  $x = a_1 \dots a_n$  is a simple chain, which means that  $G$  generates  $x$  with a single production rule  $A \rightarrow \alpha$  such that  $x \in L(\alpha)$ . By the definition of  $\delta_{\text{push}}$  and  $\delta_{\text{shift}}$ , the OPA has a support of the form (1), where  $q_0 = \langle \beta, \gamma \rangle$  and  $q_{n+1} = \langle A, \beta \rangle$ . All states  $q_i$  for  $i = 1, \dots, n$  are of the form  $\langle a'_1 \dots a'_{j_i}, \beta \rangle$ , where  $a'_1 \dots a'_{j_i}$ , with  $j_i \leq i$ , is the string obtained from  $a_1 \dots a_i$  by removing all substrings in a  $+$  operator so that they are not repeated more than once, as in the proof of Lemma 3.

For  $d > 1$ ,  $x = x_0 a_1 x_1 \dots a_n x_n$  is the body of a composed chain. Thus, if  $A \xrightarrow{*} x$  then there exists  $X_0, \dots, X_n \in V_N \cup \{\varepsilon\}$  such that  $G$  contains a production  $A \rightarrow \alpha$  with  $X_0 a_1 X_1 \dots X_n \in L(\alpha)$  and  $X_i \xrightarrow{*} x_i$  if  $x_i \neq \varepsilon$  (and  $X_i \neq \varepsilon$ ).

Thus, the OPA has a support that starts with  $\langle \beta, \gamma \rangle \xrightarrow{x_0} q'_0 \xrightarrow{a_1} \langle X_0 a_1, \beta \rangle$ , where  $q'_0 = \langle \beta, \gamma \rangle$  if  $x_0 = \varepsilon$ , and  $q'_0 = \langle X_0, \beta \rangle$  otherwise. By the inductive hypothesis,  $X_i \xrightarrow{*} x_i$  implies that there is a support  $\langle \beta_i, \gamma_i \rangle \xrightarrow{x_i} \langle X_i, \beta_i \rangle$  for each  $i = 1, \dots, n$ . We can use them to build a support for  $x$  of the form

$$\langle \beta, \gamma \rangle \xrightarrow{x_0} q'_0 \xrightarrow{a_1} \langle \beta_1, \gamma_1 \rangle \xrightarrow{x_1} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} \langle \beta_n, \gamma_n \rangle \xrightarrow{x_n} q'_n \xrightarrow{q'_0} q$$

where, for each  $i$ , either  $q'_i = \langle \beta_i, \gamma_i \rangle$  if  $x_i = \varepsilon$  or  $q'_i = \langle X_i, \beta_i \rangle$  otherwise, and  $\beta_i = X'_0 a'_1 X'_1 \dots X'_{j_i}$  with  $j_i \leq i$  is as in the proof of Lemma 3. Due to the last inductive step, we have  $q'_n = \langle X_n, X'_0 a'_1 X'_1 \dots X'_{j_n} \rangle$ . The last pop move depends on the analysis of whether  $x_0 = \varepsilon$  and/or  $x_n = \varepsilon$ , but in all four cases it leads to  $q = \langle A, \beta \rangle$ , concluding the proof.  $\square$

**Example 4.** Figure 7 displays a run of the OPA obtained from the C-OPG of Figure 4 accepting the sentence  $n + n + n/n/n + n + n$ .

The construction of a C-OPG equivalent to a given OPA is far simpler than the converse one, thanks to the explicit structure associated to words by the precedence matrix. The key difference w.r.t. the analogous construction given in [4] is that even simple chains can have unbounded length, because the  $\doteq$  relation may be circular.

First, in analogy with the definition of  $\tilde{P}$ , we define *essential supports* for both simple and composed chains, as those supports where possible cyclic behaviors of the OPA along a sequence of terminals—whether with interposing nonterminals or not—occur exactly twice.

**Definition 10.** (Essential chain supports) An essential support of a simple chain  ${}^{a_0}[a_1 a_2 \dots a_n]^{a_{n+1}}$  is any path in  $\mathcal{A}$  of the form

$$q_0 \xrightarrow{a_1} q_1 \dashrightarrow \dots \dashrightarrow q_{n-1} \xrightarrow{a_n} q_n \xrightarrow{q_0} q_{n+1} \quad (3)$$

where any cycle  $q_i a_{i+1} \dots a_{i+k} q_i$  is repeated exactly twice.

Essential supports for composed chains are defined similarly.

stack	state	current input
$\perp$	$\langle \varepsilon, \varepsilon \rangle$	$n + n + n/n/n + n + n\#$
$\perp[n, \langle \varepsilon, \varepsilon \rangle]$	$\langle n, \varepsilon \rangle$	$+n + n/n/n + n + n\#$
$\perp$	$\langle M, \varepsilon \rangle$	$+n + n/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle M+, \varepsilon \rangle$	$n + n/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][n, \langle M+, \varepsilon \rangle]$	$\langle n, M+ \rangle$	$+n/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle M, M+ \rangle$	$+n/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle \mathbf{M}+, \mathbf{M}+ \rangle$	$n/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][n, \langle M+, M+ \rangle]$	$\langle n, M+ \rangle$	$/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle D, M+ \rangle$	$/n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][/, \langle D, M+ \rangle]$	$\langle D/, M+ \rangle$	$n/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][/, \langle D, M+ \rangle][n, \langle /, D \rangle]$	$\langle n, D/ \rangle$	$/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][/, \langle D, M+ \rangle]$	$\langle F, D/ \rangle$	$/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle D, M+ \rangle$	$/n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][/, \langle D, M+ \rangle]$	$\langle D/, M+ \rangle$	$n + n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][/, \langle D, M+ \rangle][n, \langle D/, M+ \rangle]$	$\langle n, D/ \rangle$	$+n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][/, \langle D, M+ \rangle]$	$\langle F, D/ \rangle$	$+n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle M, M+ \rangle$	$+n + n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle \mathbf{M}+, \mathbf{M}+ \rangle$	$n + n\#$
$\perp[+, \langle M, \varepsilon \rangle][n, \langle M+, M+ \rangle]$	$\langle n, M+ \rangle$	$+n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle M, M+ \rangle$	$+n\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle \mathbf{M}+, \mathbf{M}+ \rangle$	$n\#$
$\perp[+, \langle M, \varepsilon \rangle][n, \langle M+, M+ \rangle]$	$\langle n, M+ \rangle$	$\#$
$\perp[+, \langle M, \varepsilon \rangle]$	$\langle M, M+ \rangle$	$\#$
$\perp$	$\langle P, \varepsilon \rangle$	$\#$

Figure 7: A run of the OPA built from the C-OPG of Figure 4 accepting the sentence  $n + n + n/n/n + n + n$ . The states truncated by erasing a repeated suffix  $\zeta$  occurring under the scope of a  $+$  operator are emphasized in boldface.

For instance, with reference to the OPA built from the C-OPG of Figure 4 an essential support of the chain  $\# [n + n + n + n + n] \#$  is:

$$\begin{aligned}
& \langle \varepsilon, \varepsilon \rangle \xrightarrow{n} \langle M, \varepsilon \rangle \xrightarrow{+} \langle M+, \varepsilon \rangle \xrightarrow{n} \langle M, M+ \rangle \xrightarrow{-+} \langle M+, M+ \rangle \xrightarrow{n} \langle M, M+ \rangle \xrightarrow{-+} \\
& \langle M+, M+ \rangle \xrightarrow{n} \langle M, M+ \rangle \xrightarrow{\langle M, \varepsilon \rangle} \langle P, \varepsilon \rangle
\end{aligned}$$

**Lemma 5.** *The essential supports of simple chains of any OPA have an effectively computable bounded length.*

**Theorem 6 (From OPAs to C-OPGs).** *For any OPA defined on an OP-alphabet  $(\Sigma, M)$ , one can compute an equivalent C-OPG from the OPA.*

PROOF. Given an OPA  $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ , we show how to build an equivalent OPG  $G$  having operator precedence matrix  $M$ . The equivalence between  $\mathcal{A}$  and  $G$  should then be rather obvious.

$G$ 's nonterminals are the 4-tuples  $(a, q, p, b) \in \Sigma \times Q \times Q \times \Sigma$ , written as  $\langle a p, q^b \rangle$ .  $G$ 's rules are built as follows:

- for every essential support of a simple chain,  $P$  contains the rule

$$\langle^{a_0} q_0, q_{n+1}^{a_{n+1}} \rangle \longrightarrow a_1 a_2 \dots a_n ;$$

where every double sequence  $a_i \dots a_k a_i \dots a_k$  is recursively replaced by  $(a_i \dots a_k)^+$  by proceeding from the innermost cycles to the outermost ones; furthermore, if  $a_0 = a_{n+1} = \#$ ,  $q_0$  is initial, and  $q_{n+1}$  is final, then  $\langle^\# q_0, q_{n+1}^\# \rangle$  is in  $S$ ;

- for every essential support

$$q_0 \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} q_1 \overset{x_1}{\rightsquigarrow} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \overset{x_n}{\rightsquigarrow} q'_n \xrightarrow{q'_0} q_{n+1}$$

of a composed chain,  $P$  contains the rule

$$\langle^{a_0} q_0, q_{n+1}^{a_{n+1}} \rangle \longrightarrow \Lambda_0 a_1 \Lambda_1 a_2 \dots a_n \Lambda_n ;$$

where, for every  $i = 0, 1, \dots, n$ ,  $\Lambda_i = \langle^{a_i} q_i, q_i^{a_{i+1}} \rangle$  if  $x_i \neq \varepsilon$ , and  $\Lambda_i = \varepsilon$  otherwise, by replacing double cyclic sequences  $\alpha_i \alpha_i$  with  $(\alpha_i)^+$  in the same way as for simple chains. Furthermore, if  $a_0 = a_{n+1} = \#$ ,  $q_0$  is initial, and  $q_{n+1}$  is final, then add  $\langle^\# q_0, q_{n+1}^\# \rangle$  to  $S$ , and, if  $\varepsilon$  is accepted by  $\mathcal{A}$ , add  $A \rightarrow \varepsilon$ ,  $A$  being a new axiom not occurring in any other rule.

Notice that the above construction is effective thanks to Lemma 5 and to the fact that subchains of composed chains are replaced by nonterminals  $\Lambda_i$ .  $\square$

**Remark 1.** The definition of C-OPG and the constructions of Theorems 2 and 6 have been given with the same approach as used in [4], i.e., avoiding possible optimizations to keep technical details as simple and essential as possible. For instance, we allowed only the  $^+$  operator in grammar's rhs as the minimum necessary extension to obtain the expressive power of OPAs: allowing, e.g., to use set union within the scope of a  $^+$  operator would have allowed in some cases more compact grammars, but such a choice would have also caused an explosion of the cardinality of the set  $\tilde{P}$ . For the same reason, we *a priori* excluded renaming and  $\varepsilon$ -rules in our grammars.

#### 4. Other equivalences among OPL formalisms

Figure 8 displays the five equivalent formalisms introduced in the literature to define OPLs. The reciprocal inclusions between MSO and OPA, between OPA and the finite equivalence classes of OPL syntactic congruence, and the inclusion of MSO in OPE have been proved without the hypothesis of non-circularity of the  $\doteq$  relation; the reciprocal inclusions between C-OPG and OPA have been restated in Section 3; it remains to consider the inclusion of OPEs in OPGs which was proved in [5] under the non-circularity hypothesis.

Let's recall from [5] that an *Operator Precedence Expression* (OPE) over  $(\Sigma, M)$  is a well-formed formula made with the characters of  $\Sigma$ ,  $\emptyset$ ,  $\varepsilon$ , the Boolean

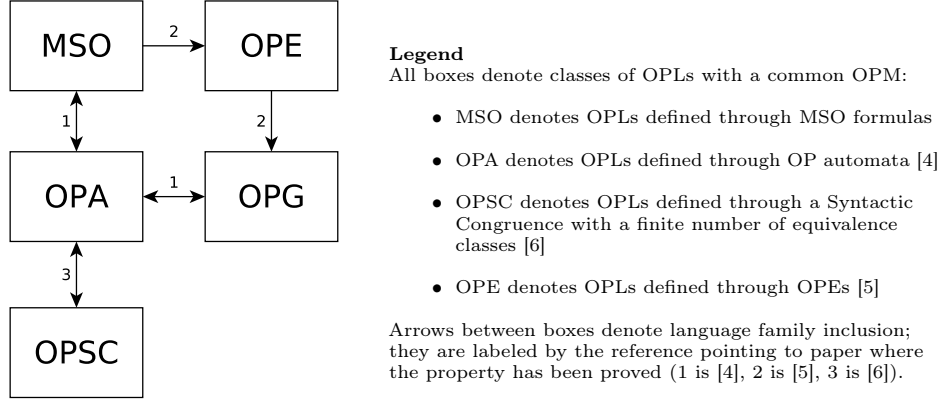


Figure 8: The relations among the various characterizations of OPLs.

operators  $\cup$ ,  $\neg$ ,  $\cap$ , the concatenation  $\cdot$ , the Kleene- $*$ , and the *fence* operation  $a[E]b$ , where  $a, b \in \Sigma_{\#}$  and  $E$  is an OPE, which “matches” two invisible parentheses between  $a$ ,  $E$ , and  $b$ , making sure that the string denoted by  $E$  corresponds to a subtree in the OPE’s ST.

For instance, let  $\Sigma$  be  $\{a, b\}$ ,  $\{a < a, a \doteq b, b > b\} \subseteq M$ . The OPE  $a[a^*b^*]b$  defines the language  $\{a^n b^n \mid n \geq 1\}$ . In fact, the fence operation imposes that any string  $x \in a^*b^*$  embedded within the context  $a, b$  be well-parenthesized according to  $M$ .

The proof of the inclusion of OPEs in OPGs used the claim that deriving an OPG from an OPE may exploit the closure properties of OPLs, in particular, w.r.t. the Kleene- $*$  operator; such a closure, however, was proved in [3] by using OPGs, again, under the non-circularity hypothesis.

Although we will see, in the next section, that all closure properties of OPLs still hold even when there are circularities in the  $\doteq$  relation, here we must consider the case of the Kleene- $*$  operator of OPEs separately. The reason is that in OPEs the Kleene- $*$  operator is now applied to subexpressions independently on the OP relations between their last and first terminal character.

Thus, it is first convenient to rewrite the OPE in a normal form using the  $^+$  operator instead of the  $*$  one to avoid having to deal explicitly with the case of the  $\varepsilon$  string. Then, subexpressions of type  $(\alpha)^+$  where the last terminal of  $\alpha$  is not in relation  $\doteq$  with the first one are replaced by the same procedure defined in [3] to prove the closure w.r.t. the Kleene- $*$  operator. The new rules will produce a right or left-linear subtree of the occurrences of  $\alpha$  depending on the OP relation between the two extreme terminals of  $\alpha$  and will avoid the use of the  $*$  and  $^+$  operators which are not permitted in the original OPGs.

The remaining substrings including the  $^+$  operator are the new rhs of the C-OPG. The other technicalities of the construction of an OPG equivalent to an OPE are identical to those given in [5] and are not repeated here.

## 5. OPLs closure properties revisited

All major closure properties of OPLs have been originally proved by referring to their generating OPGs and some of them, in particular the closure w.r.t. the Kleene-\* operator, required the  $\dot{=}$ -acyclicity hypothesis. Thus, it is necessary to prove them again. However, since some of those proofs are technically rather involved, here we simply observe that it is easier to restate the same properties by exploiting OPAs which are now fully equivalent to C-OPGs.

Observe that, in fact, closure w.r.t. Boolean operations easily follows from the determinization of nondeterministic OPAs. OPAs are closed by union, intersection, and complement, where the universe consists of an OPA accepting the *maxlanguage* of the given OPM (Definition 4). By using Theorem 2, one can convert grammars into equivalent OPAs, perform the desired Boolean operations on them, and re-convert the result into a C-OPG by Theorem 6. Similarly, the *maxgrammar* can be obtained by converting the OPA accepting the *maxlanguage* into a C-OPG.

Closure w.r.t. concatenation can be seen as a corollary of the closure proved in [4] of the concatenation between an OPL whose strings have finite length and an  $\omega$ -OPL, i.e., a language of infinite strings. The construction is based on a nondeterministic guess of the position where a string of the first language could end and a nontrivial technique to decide whether it could be accepted, even in the absence of the # delimiter. Then, the closure w.r.t. Kleene-\* is obtained simply by allowing the OPA to repeat such a guess any number of times until the real # is found.

## 6. Conclusion

We have introduced the new grammatical formalism of C-OPGs, which extends OPGs to be fully equivalent to OPAs, MSO-logic, and OPEs in the definition of OPLs. C-OPGs lift a long-standing limitation in the definability of OPLs through generative grammars, which could hamper the benefits of parallelism in parsing, as shown by recent practical applications in the field of parallel compilation [10].

C-OPGs can therefore be exploited to revisit the parallel compilation techniques of [10] (a rather unusual case where theory comes *after* its motivating application) or to improve the efficiency of techniques based on the less powerful OPGs [8]. The first steps toward this goal exploit the intuition suggested by Figure 1 and Example 3 and show that the possibility of defining data-description languages such as JSON through a flat grammar structure thanks to the  $+$  operator leads to considerable performance benefits [18].

We also mention an interesting and probably not trivial open problem with reference to the concepts of *aperiodicity*, *star-freeness*, *first-order definability*. The aperiodicity—or noncounting—property is the inability of a linguistic formalism to distinguish sentences on the basis of the number of occurrences of some subsentence; it is possessed by most, if not all, natural and programming

languages; star-freeness is the possibility of being defined by means of an expression that does not include the Kleene-\* operator; first-order definability is the possibility of defining a language by using only the first-order fragment of an MSO logic, if any.

The three concepts have been thoroughly investigated in the context of regular languages and, suprisingly, have been proved to be equivalent, i.e., to define the same subfamily of regular languages (see the classic McNaughton and Papert [19]). Attempts to generalize these equivalences to tree-languages failed dramatically [20], but succeeded in the case of OPLs [5]. However, the fairly elaborated proof of [5] relies heavily on the  $\dot{=}$ -acyclicity hypothesis, so that, at a first glance, the goal of extending it to C-OPGs looks nontrivial.

## References

- [1] R. W. Floyd, Syntactic Analysis and Operator Precedence, *J. ACM* 10 (1963) 316–333. doi:10.1145/321172.321179.
- [2] S. Crespi Reghizzi, D. Mandrioli, D. F. Martin, Algebraic Properties of Operator Precedence Languages, *Information and Control* 37 (1978) 115–133. doi:10.1016/S0019-9958(78)90474-6.
- [3] S. Crespi Reghizzi, D. Mandrioli, Operator Precedence and the Visibly Pushdown Property, *J. Comput. Syst. Sci.* 78 (2012) 1837–1867. doi:10.1016/j.jcss.2011.12.006.
- [4] V. Lonati, D. Mandrioli, F. Panella, M. Pradella, Operator precedence languages: Their automata-theoretic and logic characterization, *SIAM J. Comput.* 44 (2015) 1026–1088. doi:10.1137/140978818.
- [5] D. Mandrioli, M. Pradella, S. Crespi Reghizzi, Aperiodicity, star-freeness, and first-order definability of structured context-free languages, *Logical Methods in Computer Science* 19 (2023). doi:10.46298/lmcs-19(4:12)2023.
- [6] T. A. Henzinger, P. Kebis, N. Mazzocchi, N. E. Saraç, Regular methods for operator precedence languages, in: *ICALP 2023*, volume 261 of *LIPICs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 129:1–129:20. doi:10.4230/LIPICs.ICALP.2023.129.
- [7] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, M. Pradella, Parallel parsing of operator precedence grammars, *Inf. Process. Lett.* 113 (2013) 245–249. doi:10.1016/j.ipl.2013.01.008.
- [8] A. Barenghi, S. Crespi Reghizzi, D. Mandrioli, F. Panella, M. Pradella, Parallel parsing made practical, *Sci. Comput. Program.* 112 (2015) 195–226. doi:10.1016/j.scico.2015.09.002.

- [9] M. Chiari, D. Mandrioli, M. Pradella, Model-checking structured context-free languages, in: *CAV '21*, volume 12760 of *LNCS*, Springer, 2021, pp. 387–410. doi:10.1007/978-3-030-81688-9\_18.
- [10] L. Li, K. Taura, Associative operator precedence parsing: A method to increase data parsing parallelism, in: *HPC Asia 2023*, ACM, 2023, pp. 75–87. doi:10.1145/3578178.3578233.
- [11] M. Chiari, D. Mandrioli, M. Pradella, Cyclic operator precedence grammars for improved parallel parsing, in: *DLT'24*, volume 14791 of *LNCS*, Springer, 2024, pp. 98–113. doi:10.1007/978-3-031-66159-4\_8.
- [12] A. K. Salomaa, *Formal Languages*, Academic Press, New York, NY, 1973.
- [13] M. A. Harrison, *Introduction to Formal Language Theory*, Addison Wesley, 1978.
- [14] J. Autebert, J. Berstel, L. Boasson, Context-free languages and pushdown automata, in: *Handbook of Formal Languages (1)*, 1997, pp. 111–174. doi:10.1007/978-3-642-59136-5\_3.
- [15] D. Mandrioli, M. Pradella, Generalizing input-driven languages: Theoretical and practical benefits, *Computer Science Review* 27 (2018) 61–87. doi:10.1016/j.cosrev.2017.12.001.
- [16] D. Grune, C. J. Jacobs, *Parsing techniques: a practical guide*, Springer, New York, 2008.
- [17] S. Crespi Reghizzi, M. Pradella, Beyond operator-precedence grammars and languages, *Journal of Computer and System Sciences* 113 (2020) 18–41. doi:10.1016/j.jcss.2020.04.006.
- [18] M. Chiari, M. Giornetta, D. Mandrioli, M. Pradella, Boosting parallel parsing through cyclic operator precedence grammars, in: *SLE 2025*, ACM, 2025, pp. 44–56. doi:10.1145/3732771.3742712.
- [19] R. McNaughton, S. Papert, *Counter-free Automata*, MIT Press, Cambridge, USA, 1971.
- [20] U. Heuter, First-order properties of trees, star-free expressions, and aperiodicity, *ITA* 25 (1991) 125–145. doi:10.1051/ita/1991250201251.